🚀 JavaScript Numbers & Math Objects

(Detailed notes yet beginner friendly)

Numers in JavaScript

About Number

In JS, all types of number have same data type, i.e. number, whether it is integers (whole numbers like 10,-50) or floating point number (-0.254, 658.983)

```
let integer = 100;
let float = 99.5;
console.log(typeof integer);   // "number"
console.log(typeof float);     // "number"
```

Creating Numbers

We can create numbrs in JS by many ways:

Standard Literals:

```
let num_1 = 25;       //integer
let num_2 = 68.21     //floating-point
```

Numers in JavaScript

Exponential Notation (e): A shorthand for writing very large or very small numbers.

```
let billion = 1e9; // 1 followed by 9 zeros -> 1000000000

let tiny = 5e-6; // 5 / 10^6 -> 0.000005
```

Other Bases (Hex, Binary, Octal): You can also represent numbers in other numeral systems

```
let hex = 0xFF; // Hexadecimal (base 16) -> 255 in decimal
let binary = 0b1010; // Binary (base 2) -> 10 in decimal
let octal = 0o77; // Octal (base 8) -> 63 in decimal
```

Floating-point inaccuracy

When we add any two floating point numbers, their result is NOT totally accurate

Example → **0.2 + 0.1 = 0.30000000000000004**
and **console.log(0.1 + 0.2 === 0.3); // false**

Why this error occurs?

The numbers 0.1 and 0.2 and 0.3 have infinitely repeating representations in binary, similar to how ⅓ is 0.333... in decimal. The computer has to round them off to fit them into the 64-bit storage. These tiny rounding errors accumulate.

```
let hex = 0xFF; // Hexadecimal (base 16) -> 255 in decimal
let binary = 0b1010; // Binary (base 2) -> 10 in decimal
let octal = 0o77; // Octal (base 8) -> 63 in decimal
```

How to Fix this?

1. For financial calculations: Never use floating-point numbers. Work with integers (e.g., store money in cents).

2. For display: Use the .toFixed() method to round the result to a specific number of decimal places.

3. For comparison: Check if two numbers are "close enough" using Number.EPSILON.

Special Numeric Values

There are three values that are technically type of number

- **Infinity:** Represents a value larger than the largest possible number. It results from division by zero or exceeding Number.MAX_VALUE

```
console.log(1 / 0);                 // Infinity
console.log(-1 / 0);                // -Infinity
console.log(typeof Infinity);       // "number"
```

- **Infinity:** Represents a number smaller than the smallest possible integer

- **NaN (Not a Number):** Represents the result of an invalid or undefined mathematical operation. It's the error code for failed math

Key Property: NaN It is the only value in JavaScript that is not equal to itself.

```
console.log("hello" / 2);           // NaN
console.log(Math.sqrt(-1));         // NaN
console.log(typeof NaN);            // "number"
```

Number Properties & Methods

A. Number Constants (Limits of the Type)

- **Number.MAX_VALUE :** The largest positive number that can be represented (~1.8e+308).
- **Number.MIN_VALUE :** The smallest positive number closest to zero.
- **Number.MAX_SAFE_INTEGER :** The largest integer that can be safely represented without losing precision (2^53 - 1). **This is the one you should care about for integer math.**
- **Number.MIN_SAFE_INTEGER :** The smallest safe integer.
- **Number.EPSILON :** The smallest interval between two representable numbers. Used for floating-point comparisons.

B. Formatting and Converting Numbers

- **.toString(base) :** Converts a number to a string. You can optionally provide a base (from 2 to 36).

```
let num = 255;

console.log(num.toString());      // "255" (base 10 - default)
console.log(num.toString(16));    // "ff" (base 16 - hexadecimal)
console.log(num.toString(2));     // "11111111" (base 2 - binary)
```

Number Properties & Methods

- **.toFixed(digits) :** Formats a number to a fixed number of decimal places and returns a string. Useful for formatting currency.

```
let price = 19.991234;
console.log(price.toFixed(2));        // "19.99"
```

- **.toPrecision(digits) :** Formats a number to a specified total number of significant digits and returns a string.

```
let n = 123.456;
console.log(n.toPrecision(4));    // "123.5" (4 significant digits)
```

Math Object

What is Math Object in JS?

- JavaScript provides a built-in Math object that has properties and methods for mathematical constants and functions.
- It is not a constructor; you use it directly.

Constants (Properties) in Math Objects

- These store values.
  - **Math.PI** : Tells the value of pi (3.14159265359 approx)
  - **Math.LN2** : Tells the value of log 2 (0.30102999566 approx)
  - **Math.LN10** : Tells the value of log 2 (0.30102999566 approx)
  - **Math.E** : Tells the Euler's number (2.718281828459045 approx)
  - **Math.SQRT2** : Tells the Square Root of 2 (1.41421356237 approx)

Rounding

- **Math.round(x)** : Standard rounding to the nearest integer.
- **Math.floor(x)** : Rounds down to the nearest integer.
- **Math.ceil(x)** : Rounds up to the nearest integer.
- **Math.trunc(x)** : Removes the decimal part, leaving only the integer (truncates toward zero).

Math Object

Other Common Functions

- **Math.abs(x)** : Absolute value.
- **Math.pow(x, y)** : x to the power of y (same as x ** y).
- **Math.sqrt(x)** : Square root.
- **Math.max(a, b, c...)** : Returns the largest of the given numbers.
- **Math.min(a, b, c...)** : Returns the smallest.
- **Math.random()** : Returns a pseudo-random number between 0 (inclusive) and 1 (exclusive).

```
console.log(Math.round(4.7));          // 5
console.log(Math.floor(4.7));          // 4
console.log(Math.ceil(4.2));           // 5
console.log(Math.max(10, -5, 100, 0)); // 100
console.log(Math.min(-84, -965, -2)); //-965
console.log(Math.PI);                  //3.14159265359
console.log(Math.sqrt(4);              //2
console.log(Math.abs(-89));            //89
```

Math.random()

What is Math.random()

- JavaScript provides a built-in Math object that has properties and methods for mathematical constants and functions.
- It is not a constructor; you use it directly.
- Math.random() returns a random floating-point number between 0 (inclusive) and 1 (exclusive).
- This means it can return 0, 0.1234, 0.5, 0.99999, but it will never return 1.0.

Syntax

```
console.log(Math.random());

//It generates random number from 0 to 0.9999999 (Not 1, but 0 is included)
```

Math.random()

Generating Random number between 1 to 10

Multiply operator

```
Math.floor(Math.random()*10+1);
```

10 is here **Total Number Of Outcomes** we want (It can be anything according out choice

1 is **shift** here.
Initially it is 0.
Without 1 out number will start from 0 and end with 9.
We don't wanna that, so we added **1,** now it will shift by 1 number.
Therefore from 1 to 10

Math.floor() value for rounding down to nearest integer.
Without it our numbers will be in floating point.

Simplify the formula by using min and max. Number from 1000 to 9999

```
Math.floor (Math.random() * ( max - min + 1 )) + min

Math.floor (Math.random() * ( 9999 - 1000 +1)) + 1000
```

Explanation:
Here: max is 9999 and min is 1000.
We have added 1 here because the total numbers between 9999 and 1000 will be n-1. But we want exact n, so we added 1 to get exact total numbers between 9999 and 1000.
In this way we can get random numbers between 1000 and 9999

Why Math.random() is NOT secure ?

Why Math.random() should NOT be used for OTPs ?

When developers generates OTPs or security tokens beginners use, **Math.random()**

But this is NOT recommended method for security purposes

Let's understand Why?

1. Understanding Random Number Generation

To understand this, think of any random generator as three parts:

```
Input (Seed) → Algorithm (Black Box) → Output (Random Number)
```

Example idea:

```
Seed → Mathematical operations → Random number
```

The algorithm might:

- multiply numbers
- divide them
- apply mathematical transformations

and finally produce a number that looks random.

Why Math.random() is NOT secure ?

2. The Problem: Pseudo Random Numbers

Math.random() does not generate true randomness.
It generates Pseudo Random Numbers (PRNG).

Pseudo random means:
The numbers look random, but they are actually generated using a predictable mathematical formula.

If someone knows:
• the algorithm
• the starting value (seed)
they can reproduce the same sequence of numbers.

Example idea:
Seed = 12345

```
Algorithm → Number1 → Number2 → Number3 → Number4
```

If the attacker finds the seed, they can predict the entire sequence.

That is dangerous for:
• OTPs
• Password reset tokens
• Session IDs
• Security keys

Why Math.random() is NOT secure ?

3. Why Time Is Often Used As Input

Historically, many random generators used current time as the seed.

Example:
seed = current time in milliseconds

Why?
Because time is always changing.

But the problem is:
An attacker can guess the time range when the number was generated.

Example:
If an OTP was generated at: **10:15:30**

An attacker may try seeds around:
10:15:29
10:15:30
10:15:31

Then reproduce the same random sequence.
So time alone is not secure.

Why Math.random() is NOT secure ?

4. Real Randomness Comes From Entropy

Modern systems collect randomness from unpredictable system events.

This is called entropy.

Examples of entropy sources:

- **mouse movements**
- **keyboard typing timing**
- **network activity**
- **disk operations**
- **CPU timing variations**
- **hardware noise**
- **memory state**

All these unpredictable signals are collected by the operating system.

5. The Entropy Pool

Operating systems maintain something called an entropy pool.

It works like this:
System Events → Entropy Pool → Secure Random Generator

Many unpredictable events are mixed together, making the data extremely difficult to predict.

This entropy is then used to generate secure random numbers.

Why Math.random() is NOT secure ?

6. Cryptographically Secure Random Generators

Modern systems use something called:
CSPRNG
(Cryptographically Secure Pseudo Random Number Generator)

These generators are designed so that:

- **Even if the algorithm is known**
- **Even if some outputs are known**

The next number cannot be predicted.
This is what makes them safe for security.

7. What Is A Crypto Library?

A crypto library is a software library that provides secure tools for:

- **encryption**
- **hashing**
- **digital signatures**
- **secure random numbers**

Examples:

- **OpenSSL**
- **Node.js crypto module**
- **Web Crypto API**

These libraries use CSPRNG internally.

Why Math.random() is NOT secure ?

6. Cryptographically Secure Random Generators

Modern systems use something called:
CSPRNG
(Cryptographically Secure Pseudo Random Number Generator)

These generators are designed so that:

- **Even if the algorithm is known**
- **Even if some outputs are known**

The next number cannot be predicted.
This is what makes them safe for security.

7. What Is A Crypto Library?

A crypto library is a software library that provides secure tools for:

- **encryption**
- **hashing**
- **digital signatures**
- **secure random numbers**

Examples:

- **OpenSSL**
- **Node.js crypto module**
- **Web Crypto API**

These libraries use CSPRNG internally.

Secure Random numbers in JS

Instead of Math.random(), modern JavaScript provides secure APIs.

Browser → Use the Web Crypto API:

```
const array = new Uint32Array(1);
crypto.getRandomValues(array);

console.log(array[0]);
```

Now, in this way we can generate secure random numbers in JavaScript which are extremely difficult to predict.